A Framework for Distributed Rover Control and Three Sample Applications

Steve McGuire
Autonomy and Robotics Area
NASA Ames Research Center, MS 269-3
Moffett Field, CA, 94035-1000

Abstract

In order to develop quality control software for multiple robots, a common interface is required. By developing components in a modular fashion with well-defined boundaries, roboticists can write code to program a generic rover, and only require very simple modifications to run on any robot with a properly implemented framework. The proposed framework advances a GenericRover that could be any rover, from Real World Interface's All Terrain Robot Vehicle Jr. series, to the Fido-class rovers from the Jet Propulsion Laboratory, to any other research robot. Using these generic hardware interfaces, software designers and engineers can concentrate on the actual code, and not have to worry about hardware details. In addition to the hardware support framework, 3 sample applications have been developed to demonstrate the flexibility and extensibility of the framework.

1 Introduction

Every robotic rover architecture relies on some distributed mechanism to share rover data and to issue commands. However, these independent architectures are very hardware dependent and are not easily extensible to different rover platforms. The proposed framework allows high-level software to be written to support a reference rover. Through the use of well-defined features and interfaces, high-level controllers need not be concerned with hardware dependencies. To support this framework, a foundation layer is required to transform high-level commands into the required hardware interface for the rover.

The proposed framework was designed to support Real World Interface's (RWI) All Terrain Robot Vehicle (ATRV)-Jr. robot, but could be easily extended to any control interface. Every detail of the rover is hierarchically descended from a basic rover object. Following previous work in the group, commands may be sent to the physical hardware via standardized messages; for convenience, the K9 rover messages were used as templates. In the case of the ATRV-Jr., the hardware framework is responsible for translating these high-level messages into Common Object Request Broker Architecture (CORBA) operations. CORBA is used as an object transport by default; the commercially supported control system aboard the ATRV-Jr. communicates via CORBA. By using asynchronous command queues to process rover requests, most framework calls return to their calling function very quickly, generally only requiring a memory copy.

Distributed operation is handled transparently by the CORBA object transports, concurrently allowing multiple disparate clients to issue requests to the rover and observe the state. Much of the work of the base class involves translating robot data to appropriately standardized common output values. For example, the CORBA components for the Inertial Measurement Unit (IMU) provide one integration step of acceleration data; the framework then interpolates the second required integration to provide a rough estimate of position.

2 Hardware Support

2.1 Hardware to CORBA Interface

Because of the modular design of CORBA, certain components can be replaced with higher-performance implementations. In the ATRV Jr., the only components that are not driven by serial port are the stereo framegrabbers, which connect via two Peripheral Component Interconnect (PCI) bus slots. These framegrabbers are commercially available Hauppauge WinTV NTSC cards that utilize a Brooktree 878 chipset. The original code provided by RWI used a single-threaded, single buffering capture scheme which

did not allow the framegrabbers to function at their optimal performance level. To speed up image capture, the hardware to CORBA software layer was modified to use a framegrabber library [1] that provides multithreaded triple buffering.

The use of this library enabled a speed increase in the performance of the vision subsystem. In addition, the original software layer wasted CPU time in a compression step. Through experimentation, it was determined that the compression was wasteful in cases where the data will be used onboard or over a hard link such as wired Ethernet. The original reason to use compression was to enable image transmission over the rover's wireless Ethernet interface, which has much less bandwidth than a wired connection. Since image analysis will be used on-board, the compression was removed.

2.2 Pseudo-Subcapture

A software modification was made to the vision subsystem to enable cropping of captured images before transmission through the CORBA interface. This modification was important to speeding up image operations; by transmitting less data across layers, more CPU time can be devoted to analysis. The time required to transmit a control request is much less than that required to transmit a complete image.

Some framegrabber cards support this operation in hardware; however, the ATRV Jr.'s do not. Therefore, a software cropping solution was implemented, by which the return data buffer was filled by carefully iterating through the full-size image buffer to crop out the specified image size. The subimage is specified by using a top and left coordinate pair in conjunction with an x-length and a y-length coordinate pair.

The operation to crop an image before transmission is only slightly less computationally efficient due to the need to perform multiplication to locate indices in the source image buffer, but the reduction in transmission speed is of such great benefit so as to justify the added computation.

3 Base Calibration

3.1 Introduction

As an extension of the CORBA objects, the framework provides the ideal layer to apply correction factors to systematic rover errors and to approximate non-systematic errors. For example, using the UMBMark benchmarking tests[2], the error in the drivetrain's positional estimate can be reduced. However, the ATRV-Jr. is a skid steer robot, similar to a tracked vehicle, and depends on slipping tires to turn. Therefore, there is some undetermined error when a turn is executed that depends on the surface conditions and the requested rate of turn. So, for every surface that the robot may operate on, a new model must be developed for the rover's drivetrain.

In the course of field tests, traverse data has been collected that will allow a calibration to be approximated for the ATRV-Jr. When on an ablative surface such as gravel, the skid steer of the robot disturbs the surface. Due to the change in surface conditions, two turns of the same velocity and the same duration do not produce the same change in robot heading.

In Figure 1, the two circles are supposed to be coincident turns of the rover after a previous turn and traverse. Figure 1 is a prime example of the difficulty in calibrating a skid-steer robot. The effectiveness of the steering action is directly affected by the stability of the underlying terrain. Possible solutions include

Figure 1: Error in Turning Movement

the use of absolute positioning sensors such as the onboard electronic compass and relative sensors such as the IMU; the update speed of the compass and the inaccuracy of the IMU generally preclude their direct use

as controls on rotation and translation. The results of [5] may be useful to integrate into the base controller, as the sensors available on K9 are very similar to the sensors of the ATRV-Jr.

3.2 Resolution

The calibration setup was needed to correct for a problem that was only observed in field tests that related to the difference between a turn command and the actual result of the command. These commands were given by hand and thus were directly observable; the same class of commands is used by the sample tracker described below and thus need to become more accurate. The model of the base drivetrain that is exposed by RWI's software does not expose raw wheel counts, and appeared calibrated for non-ablative surfaces such as solid floors. On such surfaces, turns are much more accurate than for turns on gravel.

A basic calibration is proposed to determine linear coefficients of the rotational and translational velocities to characterize a surface's requirements for accurate motion. Also, because the work of [5] is available for integration into the base controller, a combination approach might give the best results.

Figure 2: CMU Omnicam aboard ATRV-Jr.

4 Sample Applications: Utilities

4.1 *grover*: A Rover Control Utility

grover was designed to exercise all of the possible functions of all of the rover extensibilities. The command line interface is suited to directing the robot through a series of low-level operations. In essence, the purpose of this application is a test and utility driver that can issue specific commands for adjusting the rover's state. For example, this utility was used in the proving tests to issue specific drive commands and to adjust the pantilt head to specific values. Also, *grover* was used to capture imagery used by E. Bandari and E. Ricks of the Autonomy and Robotics Area (ARA) to characterize the vision system aboard the rover.

4.2 *panovision*: A Telemetry and Imaging Utility

To demonstrate the flexibility of the framework, and to support other researchers in the Autonomy and Robotics Area, *panovision* was implemented. A commercially available camera was placed into an optics package manufactured by Carnegie Mellon University (CMU) to provide a full 360 degree field of view. The complete enclosure was then mounted on the ATRV-Jr using a custom manufactured mount. (Figure 2)

The chosen camera is a Sony EVI-370D block camera that has a serial interface to adjust camera parameters and a NTSC video output. By reviewing technical material provided by Sony and modifying source code provided by CMU[4], a library of camera control functions was developed. Some functions include adjustable zoom, focus control, and exposure control.

4.2.1 Telemetry Control

To facilitate other research, rover telemetry had to be captured by interrogating various hardware components within a frame of reference. This hardware requirement was accomplished by using the developed framework. In addition, assuming a proper implementation was available for another rover, only superficial modifications would be necessary to run the image capture and telemetry capture on another platform.

4.2.2 Rate of Capture

panovision was written with the express desire to capture rover telemetry at a rate sufficient to reconstruct rover operations offline. Because of the finite processor capability available onboard, image requests arrived in a command queue faster than the requests could be serviced; the result was that multiple telemetry records would be available between images. For the purposes of the other researchers in ARA, this many-to-one relationship would not be a hindrance.

5 Sample Applications: Visual Servoing

As a realistic test of the process of modifying an existing application to support the proposed framework, the tracker developed by [3] was ported to run on the ATRV-Jr. This tracker uses an in-house image class that provides many common operations such as reading, writing, and basic transformations. Generic code was available that encapsulated an image acquire operation via a custom method for each framegrabber that was to be supported. This method allows for hardware flexibility; for example, to test analysis code, the acquire might simply load an image from secondary storage.

In the operations described in [3]using the Marsokhod rover, a custom method was used to transfer an image from the hardware framegrabber buffer to a common image class. In order for the proposed framework to support this kind of synchronous operation, a support layer was added to translate the CORBA-compatible image transport into the logical memory layout required by the common image class.

5.1 Tracking Algorithm

The tracker uses the sign of the difference of Gaussian (DOG) operator in an attempt to uniquely characterize the desired target. Using a binary matching algorithm, the tracker correlates the chosen target with all possible points in the search space, assigning each possible point a score that is based on the number of matches between the chosen target and the DOG transformation of the source image. The tracker then returns the point that most accurately matches the chosen target and its associated score.

Because the target is characterized by a 32x32 window, a perfect score is one in which every pixel of the window exactly matches every pixel of the source, producing a numerical score of 1024. In actual testing, due to subtle factors, a score of over 800 indicated an appropriate match. By thresholding the best match, we can determine when the tracker has lost the target for any reason.

5.2 Template Updating

5.2.1 Appearance Correction

As the rover approaches, the initial target will change appearance; therefore, a mechanism is required to update the template image. Using a source image, the previous target and the best estimate of the current target are combined using a weighted sum. In practice, because the target represented by this weighted sum does not appear in any image, the tracker became much more sensitive to appearance changes. To make the weighted modification process work, the kernel would require updating frequently so as to guarantee an acceptable match. In contrast, when the kernel requires an update, one can replace the kernel wholesale and remove previous influences. The downside of such a replacement approach is that the target will drift over iterations of the grab-track-move cycle. In lab tests, the drift was most noticeable when tracking over relatively long distances; the drift appeared proportional to the number of iterations required.

5.2.2 Terrain Correction

To adjust for rolling terrain, one could rotate either the source image in the reverse direction, or rotate the target in the same direction. Because the target kernel is much smaller, the preferable solution is to roll the kernel. However, when a rectangular image is rotated, four black triangles will appear on the edges. Since these triangles do not appear in the source image, a false negative will be generated by the tracker, which will have lost the target. A possible solution to this problem would be to use the larger image that the kernel was first created from as the rotation source.

This solution does not work well because the roll of the rover is not about the optical axis of the camera. The roll is about the center of the mass of the rover, which is approximately 30 cm below the camera and 10 cm to the centerline of the rover. To accurately project the sensed roll onto the camera image, a coordinate transform would be necessary in addition to an image rotation. However, the rotation would be about an axis orthogonal to the image plane at a point that is not in the image. Because of the increased computational complexity involved with roll correction, the terrain is assumed to have negligible roll.

To adjust for pitching terrain, the onboard pitch sensor is utilized. In the framework, a relative tilt is defined to be the tilt angle of the pan-tilt head with respect to the rover body; an apparent tilt is defined to be the angle between the optical axis of the camera and the surrounding terrain, using a planar assumption. This apparent tilt can be calculated by summing the reading from the pitch sensor and the tilt reading from the pan-tilt head. By using the readings from the pitch sensor, the rover can compensate for uneven terrain and correct artificially high or low relative tilt angles.

5.3 Finish Conditions

Once the tracker could analyze an image and report on the best possible match of the target, the framework was utilized to center the target in the field of view and to issue a drive command to the base. In order to determine when the rover has actually reached a target, the measure of the apparent tilt angle defined above is used. Presumably, when a target is within range of an on-board instrument, the pan-tilt will be "looking" downward beyond a specific, predefined angle.

5.4 Role of Subcapture

Using the subcapture capability described above, the performance of the tracking algorithm was increased by limiting the search space. In every iteration, the rover seeks to minimize the absolute pan angle and keep the target centered in the image plane. Once the rover has met these objectives, the possible search space is limited to one quarter of the size of the initial space to increase tracking performance.

Due to the distributed nature of the vision system, a possible conflict exists where a request to change the subimage may occur, but does not take effect instantaneously. The result of such a conflict is that the returned imagery is from a different section of the image than the specified coordinates requested. To prevent this conflict, the vision object pauses the tracker until the requested image coordinates are available. The pause operation is transparent to the visual servoing client and is provided by the particular framework implementation. This operation is done at the lowest possible level to reduce the communications bandwidth required by imaging operations.

5.5 Disparity and Prediction Measure

Because the rover is moving with nearly constant translational speed, a further enhancement of the tracker would be to predict the location of the target in the next frame. By assuming a linear model of movement, the target disparity, or the distance from the target to the center point of the image, is measured and recorded. Using a simple approximation, the next location of the target in the source image is predicted. Due to insufficient resources, this information is calculated and stored for further analysis, but not integrated into the tracker control loop.

5.6 Tracker Performance

An operator selects a target using an initial image at the full working resolution of the camera. The tracker loop then drives the rover towards the target, logging an image during every iteration. During full resolution operation, the tracker loop can run at approximately 2 Hz. During subimage operation, the loop speed increases to approximately 5 Hz.

5.6.1 Laboratory

In laboratory tests, the tracker's performance is capable of driving the rover at speeds of up to 20 cm/s. See Figures 3, 4, and 5 for starting, intermediate, and ending points. The operator-designated target is symbolized by a crosshair in 3; the rover's estimates are in 4 and 5. The performance of the tracker in the lab setting is sufficiently good to warrant further investigation; however, the laboratory setting makes use of controlled lighting and a well-defined textured target, the phone directory of the Center.

Figure 3: Lab Test Begin

Figure 4: Lab Test Intermediate

Figure 5: Lab Test Complete

5.6.2 Field

In field tests, the rover's speed had to be reduced to improve the tracker's performance. Due to varying lighting as the rover approaches a target and the unevenness of the terrain, the tracker frequently was unable to follow a target during the entire traverse. See Figures 6, 7, and 8 for starting, intermediate, and ending points. Once again, the operator-designated target is denoted by a crosshair in Figure 6, and the rover's estimates are in Figures 7 and 8 Note that this traverse was the only complete traverse during field testing; other attempts failed due to factors such as appearance change and terrain condition. In Figure 8, note that the crosshair is not over the initially selected target. This disparity is due to drift in the appearance correction code.

The drift is amplified during every appearance change, and so a longer traverse will produce more drift. Eventually, the drift would have knocked the template completely off of the target; the next resulting correlation attempt would produce an element of the surrounding terrain as a result. These shortcomings point to a need to implement a more robust tracking algorithm; the control code of the ATRV-Jr. did not produce a failure point.

Figure 6: Field Test Begin

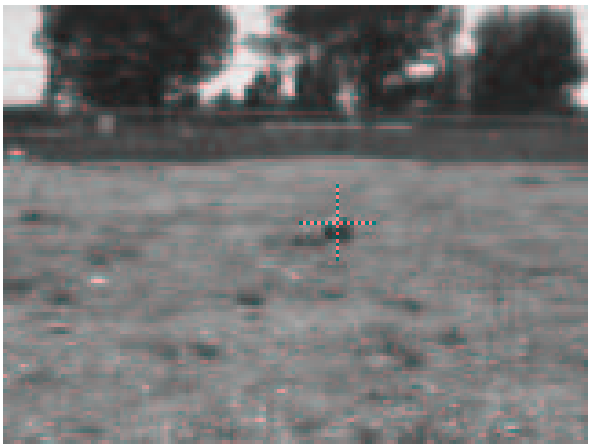

Figure 7: Field Test Intermediate

6 Summary and Conclusions

The ATRV-Jr. is envisioned to be an appropriate prototype for other inflatable-wheeled rovers; the top speed of the ATRV-Jr. is 1.5 m/s, compared to the speed of solid-wheel rovers such as the Jet Propulsion Laboratory Fido-class that have maximum safe speeds of 10 cm/s. The developed framework provides for

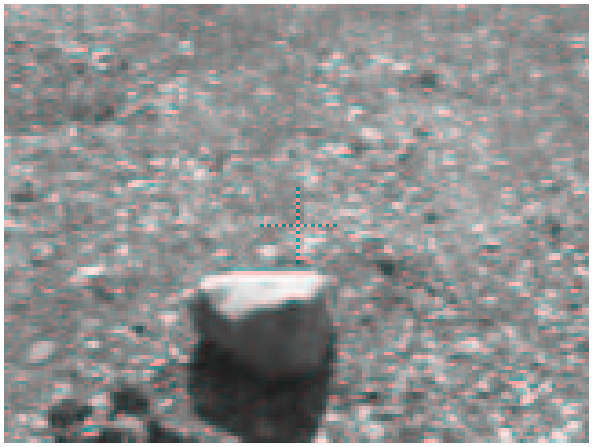

Figure 8: Field Test Complete

an extensible control and reporting interface for any type of rover that has the same logical functions. For example, most rovers have capabilities such as moving a pan-tilt head, moving the drivetrain and chassis, and retrieving current rover state.

Any software which needs to have access to rover control need only host a framework object, allowing user interface development to become independent from object transport development. Essentially, the framework establishes a local set of objects that proxy requests to the rover, hiding the underlying object transport from the developer. Through the use of such abstractions, any rover could be supported in a transparent fashion.

In addition, by using CORBA over Transmission Control Protocol/Internet Protocol (TCP/IP) as a distributed object transport, any operation can be initiated by any machine that has IP connectivity to the rover. As a side benefit, several CORBA implementations are freely available, negating the direct or hidden licensing costs associated with other distributed object models.

Many further developments are envisioned. For example, a generic simulation rover that conforms to the framework and provides mockup data and telemetry would be useful for testing rover control software offline, in a device independent fashion. Control software under development can support any rover that has basic functions; components that require specialized control such as arm control or other device control can simply extend the framework in the appropriate direction for the appropriate hardware platform. In this case, only rover-specific code would need to be written; core functionality would be provided by the framework and the framework's underlying hardware implementation.

Acknowledgments

Many thanks to M. Bualat, R. Washington, M. Deans, A. Wright. L. Edwards, K. Bass, M. Fair, E. Ricks, E. Bandari, and the other members of the Autonomy and Robotics Area of Code IC for supporting this work.

References

[1] A. Schiffler, aschiffler@home.com "libbgrab" http://www.ferzkopp.net/Software/libbgrab/

[2] J. Borenstein, L. Feng, "UMBMark - A Method for Measuring, Comparing, and Correcting Dead-reckoning Errors in Mobile Robots", UM-MEAM-94-22, University of Michigan 1994

[3] D. Wettergreen, H. Thomas, M. Bualat, "Initial Results from Vision Based Control of the Ames Marsokhod Rover", *Proceedings IROS '97*, pp. 1377-1382.

[4] J. McMahill, jmcm@frc.ri.cmu.edu "VISCA Camera Interface", Robotics Institute, Carnegie Mellon University, 1999.

[5] R. Xu, rxu@andrew.cmu.edu "State Estimation On K9", NASA Ames Research Center, 2001.